Java 103

Lesson 1

abstract "مجرد/تجريدي" ترجمة توضيحية

* يتم كتابتها بهذه الطريقة abstract قبل كلمة class

Ex: public abstract class First {

}

* معنى إنك تضع abstract قبل الكلاس ⇐ لا تريد إنشاء كائنات من هذا الكلاس

* abstract و new لا يعملان مع بعضهما أي لا يمكن إنشاء كائن (جديد) من الكلاس فانتبه لذلك.

* فائدة هذا الكلاس الذي يكون قبله abstract ⇐ يستفاد منه لأغراض الوراثة وتعدد الأشكال polymorphism

لأنه لا يمكن إنشاء كائن منه.

Lesson 2

معلومات إضافية عن abstract

* الكلاس الذي فيه abstract بالإمكان استخدامه أيضاً. جميع الدوال والمتغيرات المشتركة بين جميع الكلاسات التي ستورث من هذا الكلاس.

Lesson 3

شرح مفهوم abstract Method

* abstract Method تعتبر جزء تكاملي تضيفها لـ abstract class

* الكلاس العادي (الي من غير abstract) يسمى Concrete class

* أنواع الكلاسات ← Concrete class
abstract class ←

Concrete class ← يعني كلاس أقدر أنشئ منه كائنات
abstract class ← عكس الـ ليست كلمة في برمجة الكلاس.

* الدالة أو الـ Method التي تكون abstract تكون بلا جسم يعني بلا أقواس {} وتوضع آخر الدالة ؛
مثال: public abstract void print(); ← يعني دالة غير مطبقة غير منفذة مالها كود.

* ⇐ عشان نستخدم Abstract Method يجب أن تكون بداخل كلاس نوعه abstract

* معنى Abstract Method :- هي دالة يتم وضعها داخل abstract class ويُترك تنفيذها أو كتابتها للكلاسات التي ستورث من الكلاس الي نوعه abstract "يعني كأنه ضيفل override".

* القاعدة ببساطة هي أن جميع الدوال التي تكون من نوع abstract سيتم تطبيقها أو كتابتها في أي كلاس يرث من abstract_class First .

* الآن في الكلاس الـ Concrete والذي يرث من الـ abstract في المثال سيأتي لهفاً ، هو إنه يجب عمل override للدالة الـ abstract حتى نقدر نتعامل معها.

* فعند إعادة كتابة الدالة الـ abstract في الكلاس الـ Concrete سيتم كتابة الدالة عادي بدون كلمة abstract وتضع @override . وتكتب الدالة بالصيغة العادية.

Lesson 4

مثال توضيحي لمفهوم Abstract وكلاس Shap

شاهد الفيديو ^^

Lesson 5

استخدام abstract مع Constructor, static

* لا يمكن استخدام static مع abstract . XXX public static Abstract ---

* الـ Constructor لا يورث وإنما يُستدعى من الكلاسات الموروثة

* لا يمكن استخدام Abstract مع الـ Constructor

* الخلاصة: تذكر أن

لا يمكنك وضع abstract مع static أو مع Constructor

Lesson 6

توضيح استخدام abstract
في مفهوم تعدد الأشكال

* جاز نمثلاً وانت عملت كلاس Abstract وورثت منه كلاسات مثلاً.

الآن لو رجعت لأي Concrete class وكتبت فيه دالة بالشكل التالي :-

Abstract class

```
public void drawshap (shap s)
{
}
```

لاحظ هنا تم قبول الجملة لأنه لم نستخدم "new" يعني المفروض يرفض لأن shap نوعه abstract

⇐ ولكن هنا قوة الـ Polymorphism تكمن واضحة جيد ثم قبول الكود

لأنه يقصد بـ shap s ⇐ إنه هيمرر كائن من أحد الكلاسات اللي
ورثت shap ∴ يعني مستحيل يمرر shap لأنه Abstract

⇐ s لازم يكون أحد الكائنات من الكلاسات الموروثة من shap .

Lesson 7

توضيح استخدام abstract
في مفهوم تعدد الأشكال

Ex

```
public shape getshap()
{
    circle c1 = new circle();
    return c;
}
```

* نوع الإرجاع نوعه shape ولكن
ليس كائن من shape نفسه بل كائن من كلاس مشتق shape من

int
void زي

* الآن أصبحت الدالة getshap تطبق مفهوم الـ polymorphism في الإرجاع

```
circle C = new circle();
Rectagle R = new Rectagle();
shap S = C;
```

* ثم يعطى خطأ هنا لأنه لم نعمل new لـ shap

لكن هو كده أخذ قيمة "هو الفرق بين الإنشاء وأخذ القيمة"

* يعني كأنها كائن من shape "البوليمورفيزم" يعمل كده اليوم

Polymorphism

تأخذ أكثر من شكل ممكن دائرة أو مستطيل

new كائن لأنه معاش

R

Lesson 8

استخدام abstract

مع مفهوم تعدد الأشكال polymorphism

```
Rec r = new Rec();
```

* كده معناها إنك عملت كائن من shap

```
Shap s = r ;
```

* بل معناها إنه كائن يشير له بداخله كائن لأحد الكلاسات المشتقة من shap

```
if ( s instanceof Rectangle) //
{
```

يعني لو s نوعه Rectangle

مثال في الفيديو

```
}
```

Lesson 9

شرح الـ interface ومعنى implements

* interface: هي طريقة للتواصل بين الكائنات بصفة عامة.

* اسمح كلمة class واكتب interface

```
public class MyInterface
```

interface

* الـ interface كأنك بتكتب كلاس عادي، لكن الفرق إنه يحتوي على دوال بدون Body

كل الدوال جميعها لا تحتوي على Body

يعني ليست منفذة ليست implemented ← في الـ abstract، لكن بدون كتابة abstract.

Ex: `public void start();`

* الدوال المكتوبة بداخل كلاس interface تكون public بشكل افتراضي حتى لو لم تكتبي.
* لا يمكن إنشاء كائن من الـ interface لأنه يُعامل معاملة الـ Abstract.
* تحتوي على ثوابت final عادي جدًا
* الـ interface يحتوي على قائمة من الدوال الغير منفذة.
* كيف سيتم تنفيذ الدوال فيه بما أنه الدوال كلها بدون Body !؟

1/ عن طريق كلاس آخر يعمل implementation لهذا الـ interface.

* في كلاس التنفيذ يتم كتابته هكذا `public class App implements MyInterface{...}`

* معنى هذا الكلام: App كلاس عادي وهينفذ جميع الدوال الموجودة في الـ interface الي اسمه MyInterface

* ممكن تكتب دوال جديدة، لكن يجب عمل override للدوال الموجودة في الـ interface كلها

lesson 10

معلومات إضافية عن Interface

* الـ interface تربط بين عدة كلاسات عن طريق interface وليس عن طريق الوراثة !
* أي كلاس يعمل implements لـ interface فلازم يحتوي على الدوال الموجودة في الـ interface كلها.

lesson 11

تمرير الـ interface كبارامتر

```
public void printVal(MyInterface in)
{
    in.printData();
}
```

* عرفت كائن نوعه interface اسمها MyInterface
* ممكن نعمل كائن من الـ interface لكن ده كائن يشير لأي كائنات الكلاسات اللي implements الـ interface ستنفذ
* تعرف الـ interface في البارامتر عادي جدًا مثل أي كلاس
ولكن لا تقدر تستخدم معها new لأنها أساسًا
لا يوجد بها دوال

⇐ المعنى إنك تبعت في الدالة بارامتر نوعه interface: يعني تقدر تمرر أي كلاس يطبق هذه الـ interface كائن

* ميزة الـ interface: إنك بتستخدم الدوال الموجودة في كلاس interface بدون معرفة إيه اللي هيتنفذ
* فالمعرفة اللي سيتم تنفيذه يعتمد على الكلاس الذي سيتم تمريره.

lesson 12

مثال على استخدام الـ interface

* وعندك برنامج مثلاً ويقبل plug-in "إضافات" وتمرر له الإضافات للمطورين
مثلاً مطور هيضيف إضافة لـ pdf - تشغيل الصوت - ... الخ فكيف تدير هذا الكود لأن المطور
هيكتب الكود بتاعه هنا بقا هتستفيد من الـ interface تخلي كلاس plugin اسمه interface
وأي شخص هيعمل إضافة لازم يطبق هذه الـ interface بكل دوالها.
* لكن لن تعرف ما المكتوب داخل الدوال لأن المطور هو اللي كتبها بس هو ملزم بالدوال هذه.

lesson 13

استخدام أكثر من interface
Implement Multi-Interfaces

Ex: public class Demo implements X, Y, Z { }

حيث X, Y, Z عبارة عن interfaces.

⇐ المعنى كده أن جميع الدوال X, Y, Z بتاعتهم لازم تُنفذ داخل الكلاس Demo

⇐ تم عمل هذه الحركة حل مشكلة Multi_Inheretance " C++ تدعم تعدد الوراثة بينما جافا لا تدعم " لكن بهذه الحركة حيلة.

⇐ هام: عزيزي الفيديو " Demo بنفيذ أكثر من كلاس " Demo, Z, Y, X

---

lesson 14

عمل الـ extends لـ interface

+ هام شاهد الفيديو

Ex: public interface MyInterface2 extends MyInterface
{ public void justLoad();
}

الـ interface مع الـ interface يعمل وراثة.

⇐ الـ interface مع الـ interface extends.

اذن ⇐ لما نيجي نطبق interface على كلاس نقول implements.

⇐ الكلاس مع الكلاس extends.

~~⇐ الـ interface مع الكلاس extends.~~

⇐ الكلاس مع الـ interface implements

* الآن في الـ interface اللي اسمها MyInterface2 جوه كلاس MyInterface ⇐ 1 من MyInterface2

Lesson 15 — extends و implements

```
public class Adv extends First implements MyInterface, plugin
{
}
```

* بالإمكان إنك تطبق الوراثة والـ implements لـ interfaces
في سطر واحد

* الآن الكلاس Adv يرث من First ويطبق الـ MyInterface وكمان plugin

```
Advance a = new advanced();
```

* الآن الكائن a نوعه Advance حتى نوعه First لأنه يرث منه ونوعه MyInterface
وكمان نوعه plugin « معنى عام »

* حتى a كائن يرث من First ويطبق الاثنين الـ interfaces

Lesson 16 — Anonymous Class واستخدامه مع الـ interface

* طبعاً هذا شيء يستخدم في الأنظمة نسبة وتناسب البيانات .

Ex
```
First f = new First();
App a = new App();
f.printVal(a);
```

a من App وبما أنه ينفذ الـ MyInterface فسيتم تمريره عادي
لأن printVal يستقبل MyInterface

output : welcome to app

```
First f = new First();
MyInterface i = new MyInterface() { الدوال التي سيتم تنفيذها من الكلاس الذي يطبق الـ MyInterface } ;
```
← : syntax

* وهذا ليس معناه إنك عملت كائن من MyInterface

* بالإمكان عمل تطبيق لـ interface بهذه الصيغة بدلاً من إنشاء كلاس جديد وعمل implements .

* معنى الـ syntax : أنشئ كائن جديد من هذا الكلاس "يجب أن يكون الكلاس يطبق الـ interface" ولازم
يتم وضع دوال الـ interface بداخل الأقواس {=}

* التفسير: كأنك عملت كلاس جديد بدون اسم «Anonymous class» وأنشأت منه كائن.

* شاهد الفيديو هتفهم أكتر.

---

lesson 17

Anonymous object كبارامتر

هنستخدمه كتير جدًا في الأندرويد والـ Gui

* كما في الدرس السابق الكلاس اللي من غير اسم تكوينه الـ syntax كما يلي:

```
MyInterface i = new MyInterface(){ ≡ };
```

① الكلاس الوهمي
② انشاء منه كائن
③ يخزن في i

كلاس جديد بدون اسم وانشاء منه كائن

④ يتم استدعاء الدالة العادية ومرر i مثل: `F.printVal(i);`

كائن من كلاس First تم تعريفه

دالة تأخذ بارامتر نوعه MyInterface

الجديد:

* جدًا نقيل sytax معقدة جدًا

* الـ syntax الجديد أن أثناء تمرير الباراميتر `F.printVal( ( ) );`

يقبل أي كائن من MyInterface

Syntax:

```
F.printVal( new MyInterface(){ ≡ } );
```

كلاس بدون اسم ← كائن بدون اسم

الدوال التي سينفذها الكلاس الذي سيطبق implements الـ MyInterface

⟸ هنستخدم جدًا هنتعامل معاه بكثرة في الأندرويد والـ Gui.

lesson 18 — الكلاسات و Anonymous object

- new My Interface () {...};
  الـ interface

* هنا أنشئت كائن بدون اسم
وكذلك كلاس بدون اسم

Example in video..

```
new MyInterface(){
   public void printData(){ system.out.println("anan"); }
}.printData() ;
```

لاستخدام هذا الكلاس أو الكائن في نهاية {} الدالة يتم وضع نقطة ثم اسم الدالة

الشرح :- أنشئ كائن جديد بدون اسم من الكلاس الذي بدون اسم ثم بعد الإنشاء استدعي منه دالة printData

---

lesson 19 — "تحويل الأنواع" Type Casting

* يعني تحول من نوع بيانات إلى آخر

```
int x = 10 ;
int y = 20.44 ;

int z = x ;
```
← لأن z و x نوعهم int

~~int z = y ;~~ X X خطأ لأنه مختلفين في النوع

```
int z = (int) y ;
```
← صيغة التحويل

(int) الأولى: أعلى / الثانية: أحول إلى

← يعني خذ قيمة y وحولها لـ int وخزنها في z

* يمكن استخدام التحويل أيضاً مع الكائنات

* التحويل للكائنات شاهد فائدته في الفيديو ، هام

lesson 20

معلومات إضافية عن
Type casting

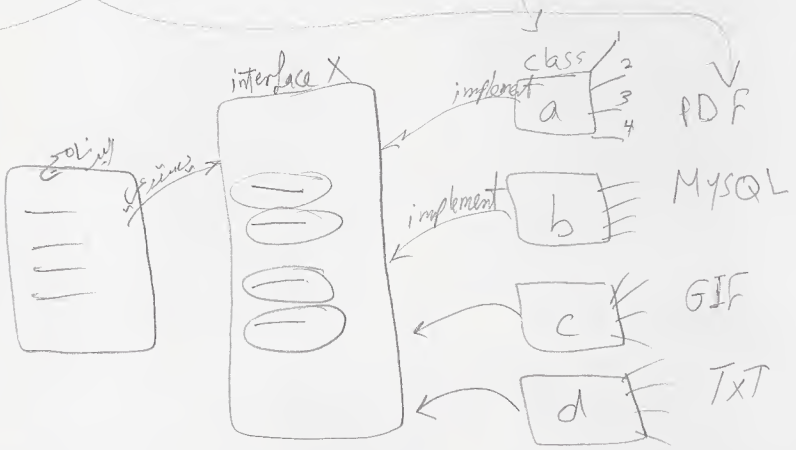
+ شاهد الفيديو

---

lesson 21

شرح تفصيلي لـ Interface

كل واحد من هذه الكلاسات يطبق الـ interface بطريقة مختلفة لكن مع وجود نفس الدوال الأربعة.



---

lesson 22

شرح الـ Empty Interface

* لا يستخدم الكلاسات التي تطبق الـ interface في كل مكان يتم عمل Empty Interface وجعل الكلاسات المطلوبة تـ implements الـ interface . ومن ثم نمرر لأي كلاس عادي ونعمل دالة مثلاً:

```
public void (Interface i)
{
}
```

هنا البرامتر هو أي كائن من الكلاسات التي تطبق الـ interface

Ex2:

```
public EInterface getIt()
{ A z = new A();
} return z;
```

لازم يرجع كائن من نوع كلاس يطبق EInterface وهو A

lesson 23 — مثال لشرح فائدة الـ interface الـ plugin

هام

---

lesson 24

ما هو enum

* اختصاراً لـ enumeration «التعداد»

+ التعريف: هي عبارة عن طريقة لجمع ثوابت لها علاقة ببعضها البعض في شيء واحد
  ← تقدر تستخدمها في أي مكان
  ← الثوابت تكون محددة بعدد معين

* enum: هي طريقة تستخدمها إنك تضع بداخلها ثوابت.
وهذه الثوابت تستخدمها في أي مكان وهذه الثوابت تكون محددة بعدد معين
ومثلاً لها علاقة ببعضها البعض مثال: أيام الأسبوع = ٧ أيام ثابتة مفيش غيرها
ولها أسماء محددة - مثال آخر: الجنس: ذكر/أنثى ---

```
package x;
public enum Gender {
    MALE , FEMALE ;
}
```

أنشأنا enum
وضعنا بداخله الثوابت
والثوابت كالعادة نعرفها بس نكتب بحروف كبيرة

⇐ الآن أنشأنا ثابتين MALE و FEMALE

⇐ أي ثوابت بداخل enum تعتبر static و final افتراضياً ولا يمكن تغييرها

```
system.out.println(Gender.MALE);
```

output : MALE

+ enum: أي حاجة تتكد ومحددة ← أيام الأسبوع - الجنس - شهور السنة ....

* بإمكانه كتابة متغيرات ودوال عادية داخل enum زي الكلاس.

lesson25 استخدام الـ enum مع parameters

+ الثوابت يفصل مابينها فاصلة عادية , وفي الأخر فاصلة منقوطة ;

* قيم الثوابت توضع بين الأقواس

Ex: MALE (10) , FEMALE (20) ;

* القيم كلها لازم تكون من نوع واحد مثلاً int

private int val;

private Gender(int x)
{ this.val = x;
}

+ الفائدة : إذا كان للثابت قيمة ⇐ لازم تعرف له متغير داخل الـ enum

* هذا المتغير هيحتفظ بالقيمة اللي بين الأقواس .

* وبعدد الثوابت سيكون هناك نسخ من المتغير val

يعني مثلاً ثابتين ⇐ لما تستدعي الثابت الأول ستخزن القيمة ١٠ في val
ولما يستدعى الثابت الثاني سيكون نسخة خاصة من val وبخزن فيها ٢٠
يعني كل واحد منهم سيكون له نسخة val خاصة فيه

Gender.123 ←

* الـ enum لا تقدر تنشأ من كائنات لأن الإنشاء أوتوماتيك

* يجب عمل Constractor داخل enum لو هتضع قيم للثوابت ويكون هذا الـ Constractor ⇐ private والسبب

* البارامتر الخاص بالـ Constractor لازم يتوافق مع النوع الذي تم وضع القيم فيه بتاعة الثوابت

* الآن لو أنت استدعيت MALE(10) ⇐ كأنك استدعيت الـ Constractor بـ ١٠

* بالإمكان وضع أكثر من قيمة مثل MALE(10, "male Aman") ويكون هنا يجب إنشاء متغيرين واحد نوعه int والآخر string , كذلك في حالة الاستدعاء والإنشاء لـ Constractor .

* في حالة وضع المتغيرات private يجب عمل دوال الـ Getter & setter .

lesson 26 — استخدام ال Enumerator

* طريقة استدعاء القيم من الثوابت

```
Gender.MALE.sVal
```

(اسم الكلاس — الثابت — القيم الموجودة داخل MALE)

* enum طريقة مريحة لعمل الثوابت
* شاهد الفيديو للتوضيح

---

lesson 27 — استخدام ال Enumerator بداخل الكلاس

* بالإمكان كتابة enum بداخل الكلاس ويعتبر جزء من أجزاءه.

```
public class A {
  public int x;
  public enum Gender{
      MALE, FEMALE;
  }
}
```

* وزي الدرس السابق بالضبط

---

lesson 28 — استخدام ال Enum مع ال Method

* أي ثابت تم تعريفه مثلاً :- MALE(10,"Hi") - يعتبر كأنه كائن نوعه Gender مثلاً ع حسب التسمية.
* وبما أنه كائن فمعناها إنه يمتلك جميع الأشياء التي تعرفها داخل enum
* يعني أي شيء ستعرفه داخل ال enum ⟸ يمتلكه كل ثابت
* يعني لما تكتب أي دالة يمكن استدعائها عادي نفرض كلاس enum اسمه Gender وبه ثابت MALE ودالة print

```
∴ Gender.MALE.print();
```

lesson 29 — استخدام enum مع Abstract

* الـ enum يقبل بداخله دوال Abstract "Abstract Method"

* أي دالة abstract موجودة بداخل enum ⇐ لازم كل ثابت موجود بداخل enum يعمله @override

```
public enum Gender {
    MALE(10,"welcom")
    { @override
      public void printString()
      { system.out.println("Male");
      }
    }
```

تكملة الكود "المتغير ح والـ Constractor"

```
public abstract void printString();
```

---

lesson 30 — استخدام enum مع switch

* من مميزات enum انك تقدر تستخدمها مع switch

Gender x = Gender.FEMALE ⟶ يعني x قيمته FEMALE

```
switch(X)
{ case Female:
      system.out.print"Hi";
      break;
}
```

* يعطي البرنامج ميزة القابلية للقراءة والترتيب

---

lesson 31 — استخدام enum مع مصفوفة values

* لو عندك كلاس اسمه Gender ونوعه enum وجيت تكتب

Gender. [دوال]

لمولىس به دوال

س/ من اين أتت الدوال مع العلم انك لم تكتب دالة؟

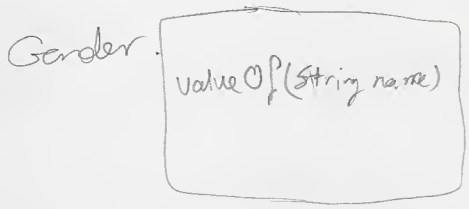
ج/ الـ Compiler تاع جافا لما انت بتنشأ أي enum بيعمل مصفوفة اسمها Values() تحتوي ع الثوابت اليانت عملتها ومرتبة حسب الترتيب اليانت عملته

Ex: Gender.Values() → دي تعتبر مصفوفة

```
for(Gender g : Gender.values())
{   g.printString();
}
```

* راجع الدرس ٨٧ جافا ١٠١ عشان الصيغة بتاعت for

lesson 32 — استخدام enum و valueOf



* valueOf : الدالة دي بتطبيعوا لاسم الثابت
ع شكل نص وترجعلك الثابت الفعلي
"عين الكائن نفسه"

Ex

```
Gender g = Gender.valueOf("welcome");
```

* يعني هي طريقة للحول ع الثابت بشكل نصي

* شاهد الفيديو ⇐ يتشرح باستخدام try , catch عشان لو المستخدم أدخل اسم غير موجود

---

lesson 33 — العلاقة بين enum , interface

* الـ enum يقدر يعمل implements لـ interface

* شاهد الفيديو

---

lesson 34 — خصائص الـ enum

* بإمكانه حفظ الميزات من الـ enum مده. مش غلط

* لكن يفضل وضعها عشان تحتفظ بالقيمة اي بايت عن طريق الواجب ، وتقدر توصلها عن طريق جميع الدوال.

---

lesson 35 — الكلاس الاسمه object وفائدته

هاد

* يوجد كلاس في جافا اسمه object وهو الكلاس الأب لجميع الكلاسات في جافا

* يعني أي كلاس تعمله بيكون تلقائياً مشتق من object

* trick :- هو كلاس ويمكن اعمله object

* فائدته انك لو عايز تمرر أي كائن من أي كلاس فستخدم object لأن جميع الكلاسات مشتقة منه تلقائياً .

lesson 36 — استخدام الكلاس object كبارامتر

* بيفنع عادي ^^
* شوف الفيديو

---

lesson 37 — استخدام كلاس object كـ Return value

* شاهد الفيديو

---

lesson 38 — جلب معلومات باستخدام getclass

* يوجد في جافا كلاس يسمى Class

class | Class { }

* كل الدوال ترث دالة اسمها getclass فيها معلومات عن الكائن

```
A val = new A();
val.getclass().getName();
```

getName() → دالة بتطلع اسم الكلاس

getclass() → Return the runtime class of an object

يعني بيرجع كائن نوعه كلاس

---

lesson 39 — استخدام الدالة ككائن أثناء الإرجاع

Class B

```
public A getobj()
{ A x = new A()
  return x;
}
```

Main

```
B val = new B();
A y = val.getobj();
y.x = 10;
val.getobj().x = 10;
```

A y → ودي كلام

y.x = 10; → ده مسموح

val.getobj().x = 10; → البديل

- في جافا يوجد syntax يمكنك إنك تتعامل مع القيمة الراجعة أو الكائن الراجع مباشرة بدون تخزينه

* كل دالة تمثل الكائن اللي بيرجعه

Ex: Method.Method.Method.....

lesson 40

(استخدام instanceOf للتعرف على نوع الكائن)

* إذا عندك دالة تأخذ بارامتر نوعه object وطبعاً أكثر من كائن مختلفين الكائنات عن بعضها وعشان تعرف الكلاس بتاعه ⇐ استخدم instanceOf
* فائدتها ممكن تعمل casting عشان تعرف تستخدم إي إنت عايز.
* شاهد الفيديو

---

lesson 41

مقدمة في الـ Nested Classes

* معناها: كلاسات بداخل بعضها البعض
* بإمكان إنشاء كلاسات بداخل أي كلاس
* الكلاس الأساسي يسمى outerclass والكلاس المُنشأ داخله يسمى innerclass

---

lesson 42

أنواع الـ Nested classes

* الكلاس الداخلي يأتي بنوعين: ① static ② non-static "Inner."
* في العادة يتم تسمية الـ non-static بـ Innerclass.
* الكلاس الـ static يتم كتابته هكذا ⇐ static class Ex{=}
* ... ⇐ public static class Ex{=} مثل Access Modif. الـ public private يتم وضع أحد الـ not static

---

lesson 43

Static Nested Class

* الكلاس الذي نوعه static ما يقدر يتعامل مع الأشياء التي في الكلاس الخارجي والتي تكون نوعها static
* الكلاس الذي نوعه static لا يقدر يوصل للمتغيرات والدوال التي من نوع غير static
* لتعريف كائن لديمن الكلاس الداخلي: A.كلاس الداخلي Ex = new A.كلاس الداخلي();

مثال: A.X theValue = new A.X();

lesson 44 — التعرف على Inner classes

* لا يمكن تعريف متغيرات ودوال من نوع static داخل الـ Inner Class العادي

* يمكن تعريف ثابت من نوع final بداخل Inner class عادي ⇐ public final static int y=10;

* الـ Inner class يقدر يوصل لأي عناصر في الـ outer class حتى لو كانت private وحتى لو كانت static أو غير static

* لا يمكن إنشاء كائن من هذا النوع مباشرة بل لابد من إنشاء كائن من الكلاس الأساسي outer class أولاً ثم يتم إنشاء كائن من Inner class عن طريق كائن الـ outter class

* مثال:

```
A val = new A();
A.Y val2 = val.new Y();
```

* شاهد الفيديو

---

lesson 45 — Local Inner class

* في جافا يمكن تعريف كلاس بداخل دالة، وهذا يسمى Local Inner class

* سمي كلاس محلي ⇐ لأنه لا يمكن إنشاء منه كائنات إلا بداخل الدالة المعرف فيها فقط، يعني معروف داخل هذه الدالة فقط

* شاهد الفيديو

* لا يمكن استخدام Access Modifiers مع الـ Local Class لأنه معرف داخل دالة، وملوش أي صلة بالكلاس الخارجي، يعني هو داخل الدالة فقط outter class

lesson 46

Anonymous Inner Class

+ عند كتابة كلاس جديد بداخل دالة هكذا

```
public void anotherRun() {
    new A() {};
}
```

دي معناها إنك عملت extends لـ A والدوال الجديدة هتكتبها بين القوسين

+ شاهد الفيديو

ختام الدورة ^^